ParaWise/CAPO Parallelization Environment

Henry Jin

hjin@nas.nasa.gov

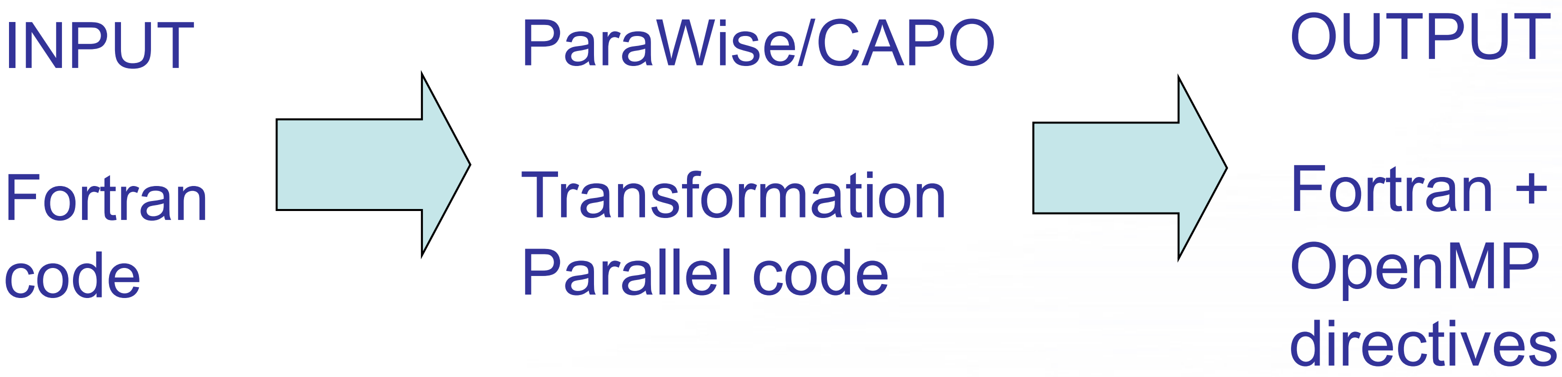
NASA Advanced Supercomputing (NAS) Division

NASA Ames Research Center

June 13, 2005

Key Ideas

- Interactive environment for semi-automatic parallelization of Fortran application codes
- Generated codes in recognizable form by user



INPUT

Fortran code

ParaWise/CAPO

Transformation Parallel code

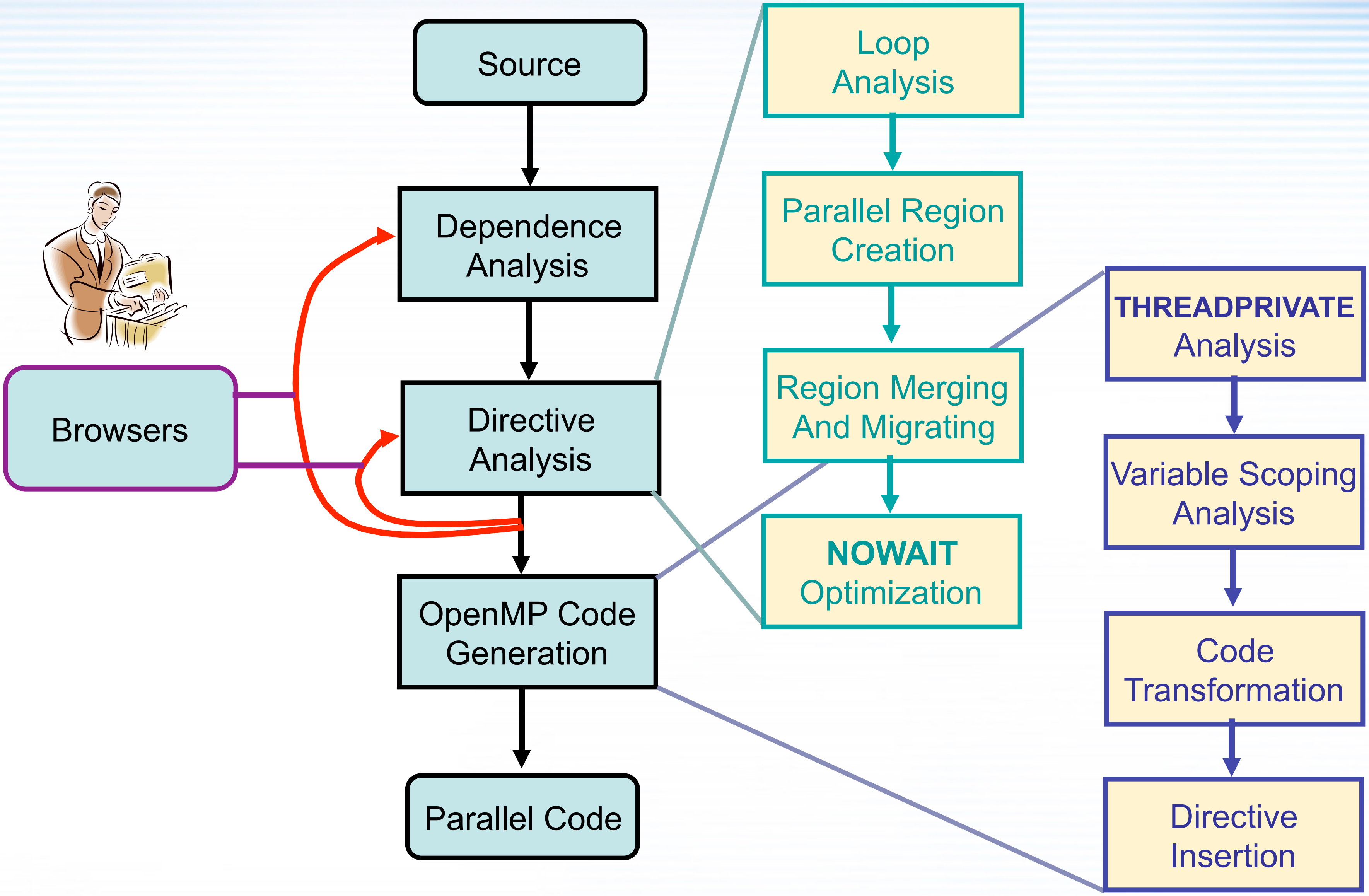
OUTPUT

Fortran + OpenMP directives

ParaWise and CAPO

- ParaWise
  - Semi-automatic, developed by Parallel Software Products
  - Accurate symbolic, value based, interprocedural data dependence analysis
  - Domain decomposition for generating message-passing codes
  - A set of browsers for user to interact with the parallelization process
- CAPO
  - A module for generating OpenMP parallel codes, developed at NASA Ames
  - Exploits loop-level parallelism
  - Directives browsers to guide the parallelization process
  - Currently integrated with ParaWise

Interactive Parallelization Process



Generation of OpenMP Code

- Identify parallel loops, including loops for setting up possible pipeline
- Construct parallel regions from parallel loops
- Merge consecutive parallel regions and migrate parallel regions as high as possible in the call path
- Perform NOWAIT optimization for consecutive parallel loops inside a parallel region
- Automatically identify and define variable scopes, such as **`SHARED`**, **`PRIVATE`** and **`REDUCTION`**
- Detect and produce **`THREADPRIVATE`** directives for common blocks

Code Generation Process

serial code

```
do K=
  ...
end do
call subwork
...

subroutine subwork
do J=
  ...
end do
do J=
  ...
end do
return
end
```

```
!$OMP PARALLEL DO

   do K=
     ...
   end do
!$OMP END PARALLEL DO
   call subwork


   ...


   subroutine subwork

!$OMP PARALLEL DO
   do J=
     ...
   end do
!$OMP END PARALLEL DO
!$OMP PARALLEL DO
   do J=
     ...
   end do
!$OMP END PARALLEL DO

   return
   end
```

identify parallel loops
create parallel regions

Code Generation Process (cont.)

```
!$OMP PARALLEL DO

   do K=
     ...
   end do
!$OMP END PARALLEL DO
   call subwork

   ...

   subroutine subwork
!$OMP PARALLEL
!$OMP DO
   do J=
     ...
   end do
!$OMP END DO
!$OMP DO
   do J=
     ...
   end do
!$OMP END DO
!$OMP END PARALLEL
   return
   end
```

merge parallel regions

```
!$OMP PARALLEL
!$OMP DO
   do K=
     ...
   end do
!$OMP END DO
   call subwork
!$OMP END PARALLEL
   ...

   subroutine subwork

!$OMP DO
   do J=
     ...
   end do
!$OMP END DO NOWAIT
!$OMP DO
   do J=
     ...
   end do
!$OMP END DO NOWAIT

   return
   end
```

migrate parallel regions
generate NOWAIT

Automatic Code Transformation

- Privatization of common block variables
  - if cannot be handled with **THREADPRIVATE**
- Routine duplication
  - to resolve conflicts of usage
- Reduction on an array variable
  - update local variable in parallel, then the shared array variable in a critical region
- F90 array syntax to loop nest
  - so that **OMP DO** can be applied
- Loop interchange
  - for better cache utilization

Routine Duplication

- Call inside a parallel region, but not inside a parallel DO

inside parallel region

outside parallel region

```
   call sub
   do K=
     ...
   end do
   ...
   call sub

   subroutine sub
   do J=
     ...
   end do
```

```
!$OMP PARALLEL
   call sub
!$OMP DO
   do K=
     ...
   end do
!$OMP END PARALLEL
   ...
   call sub

   subroutine sub
!$OMP PARALLEL DO
   do J=
     ...
   end do
```

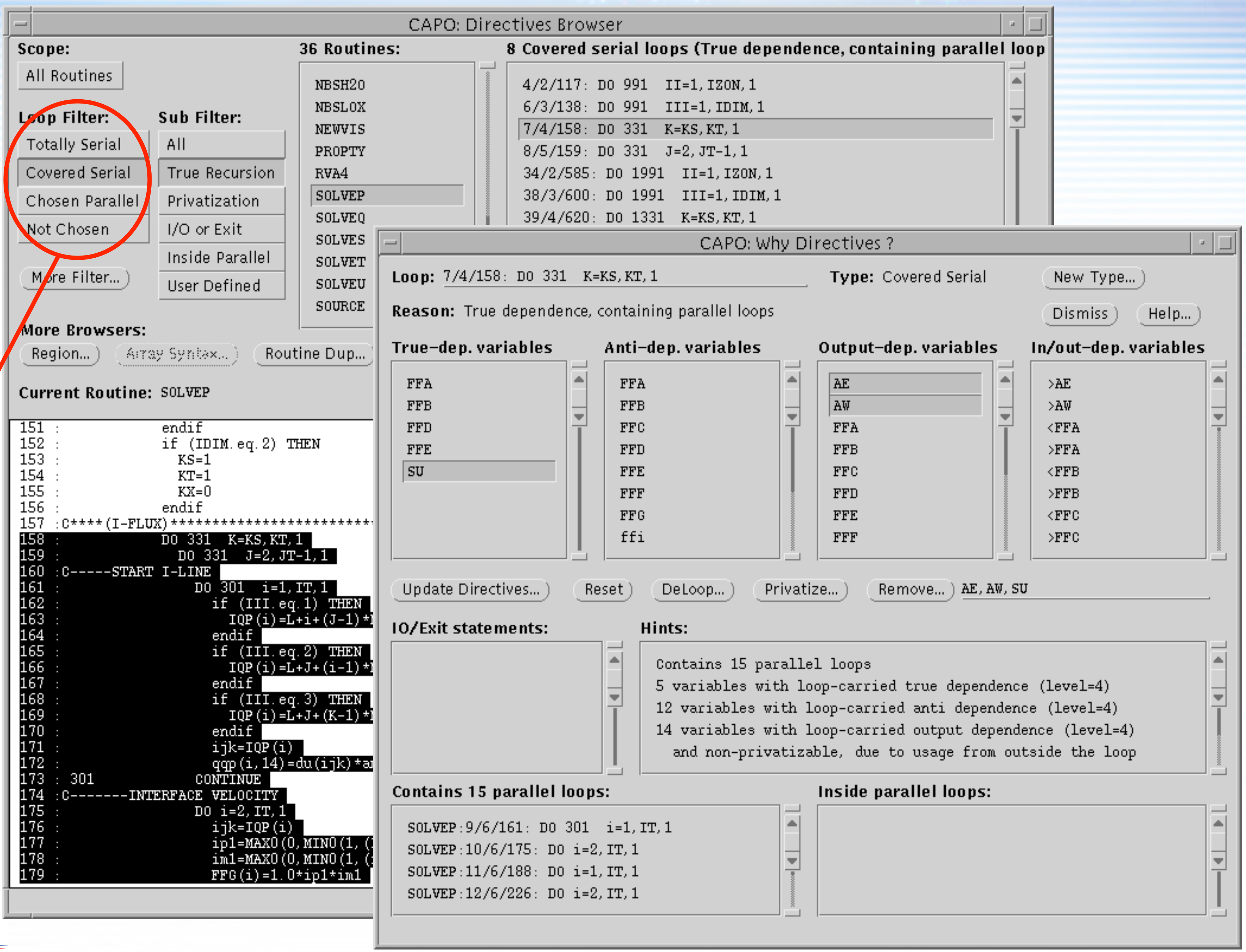
```
!$OMP PARALLEL
   call cap_sub
!$OMP DO
   do K=
     ...
   end do
!$OMP END PARALLEL
   ...
   call sub

   subroutine sub
!$OMP PARALLEL DO
   do J=
     ...
   end do

   subroutine cap_sub
!$OMP DO
   do J=
     ...
   end do
```

Identifying Parallel Loops - The Key Issue

- *Code developers want to*
  - find all the loops that can be parallelized
  - find all those that look ‘serial’
  - find which of the ‘serial’ don’t affect parallel performance and which are critical
  - fix the code so that the critical ‘serial’ loops can be parallelized

- *CAPO enables this function by*
  - categorizing different loop types
  - solving through user interaction
  - generating parallel code with directives automatically

Directives Browser Window



CAPO: Directives Browser
Region... Array Syntax... Routine Dup...
More Filter...
CAPO: Why Directives ?
New Type...
Dismiss
Help...
Update Directives... Reset DeLoop... Privatize... Remove...

Scope: All Routines

Loop Filter: Totally Serial / Covered Serial / Chosen Parallel / Not Chosen

Sub Filter: All / True Recursion / Privatization / I/O or Exit / Inside Parallel / User Defined

More Browsers:

36 Routines: NBSH2O, NBSLOX, NEWVIS, PROPTY, RVA4, SOLVEP, SOLVEQ, SOLVES, SOLVET, SOLVEU, SOURCE

8 Covered serial loops (True dependence, containing parallel loop

```
4/2/117:  DO 991   II=1,IZON,1
6/3/138:  DO 991   III=1,IDIM,1
7/4/158:  DO 331   K=KS,KT,1
8/5/159:  DO 331   J=2,JT-1,1
34/2/585:  DO 1991   II=1,IZON,1
38/3/600:  DO 1991   III=1,IDIM,1
39/4/620:  DO 1331   K=KS,KT,1
```

Current Routine: SOLVEP

```
                 endif
                 if (IDIM.eq.2) THEN
                   KS=1
                   KT=1
                   KX=0
                 endif
C****(I-FLUX)*************************
                 DO 331  K=KS,KT,1
                   DO 331  J=2,JT-1,1
C-----START I-LINE
                     DO 301  i=1,IT,1
                       if (III.eq.1) THEN
                         IQP(i)=L+i+(J-1)*
                       endif
                       if (III.eq.2) THEN
                         IQP(i)=L+J+(i-1)*
                       endif
                       if (III.eq.3) THEN
                         IQP(i)=L+J+(K-1)*
                       endif
                       ijk=IQP(i)
                       qqp(i,14)=du(ijk)*a
 301                 CONTINUE
C-------INTERFACE VELOCITY
                     DO i=2,IT,1
                       ijk=IQP(i)
                       ip1=MAX0(0,MIN0(1,(
                       im1=MAX0(0,MIN0(1,(
                       FFG(i)=1.0*ip1*im1
```

Loop: 7/4/158: DO 331   K=KS,KT,1

Type: Covered Serial

Reason: True dependence, containing parallel loops

True-dep. variables	Anti-dep. variables	Output-dep. variables	In/out-dep. variables
FFA	FFA	AE	>AE
FFB	FFB	AW	>AW
FFD	FFC	FFA	<FFA
FFE	FFD	FFB	>FFA
SU	FFE	FFC	<FFB
	FFF	FFD	>FFB
	FFG	FFE	<FFC
	ffi	FFF	>FFC

AE, AW, SU

IO/Exit statements:

Hints:

```
Contains 15 parallel loops
5 variables with loop-carried true dependence (level=4)
12 variables with loop-carried anti dependence (level=4)
14 variables with loop-carried output dependence (level=4)
  and non-privatizable, due to usage from outside the loop
```

Contains 15 parallel loops:

```
SOLVEP:9/6/161:  DO 301   i=1,IT,1
SOLVEP:10/6/175:  DO i=2,IT,1
SOLVEP:11/6/188:  DO i=1,IT,1
SOLVEP:12/6/226:  DO i=2,IT,1
```

Inside parallel loops:

loop type

Loop Types Identified with Directives Browser

Totally Serial

Problem : Potentially severe

- Serial due to loop-carried true dependence present and/or,
- Serial due to loop-carried pseudo (memory re-use) dependence by a non-privatizable variable
- Not contained in, or containing ANY parallel loops - entirely serial
- Sequential execution can prevent effective parallel performance

Possible Solutions :

- True dependence may have been assumed, may be proven to no longer exist if user knowledge is added.
- Investigate loop-carried pseudo dependence - add user knowledge to prove non-existence.
- Investigate privatization preventing true dependences from/to outside of loop - add user knowledge to prove non-existence

Browser shows serializing dependences (textually and graphically)

Loop Types Identified with Directives Browser (cont.)

Covered Serial

Problem: May be important

- Also a serial loop, but contains or is contained in a parallel loop so some parallelism will be exploited.
- If contains parallel loops, parallel performance can be enhanced by parallelism at this higher level.

Possible solutions :

- Can be treated in a similar manner to the “serial” loop type described previously.

Browser shows serializing dependences and surrounding parallel loop(s) and/or contained parallel loops

Loop Types Identified with Directives Browser (cont.)

Chosen Parallel :

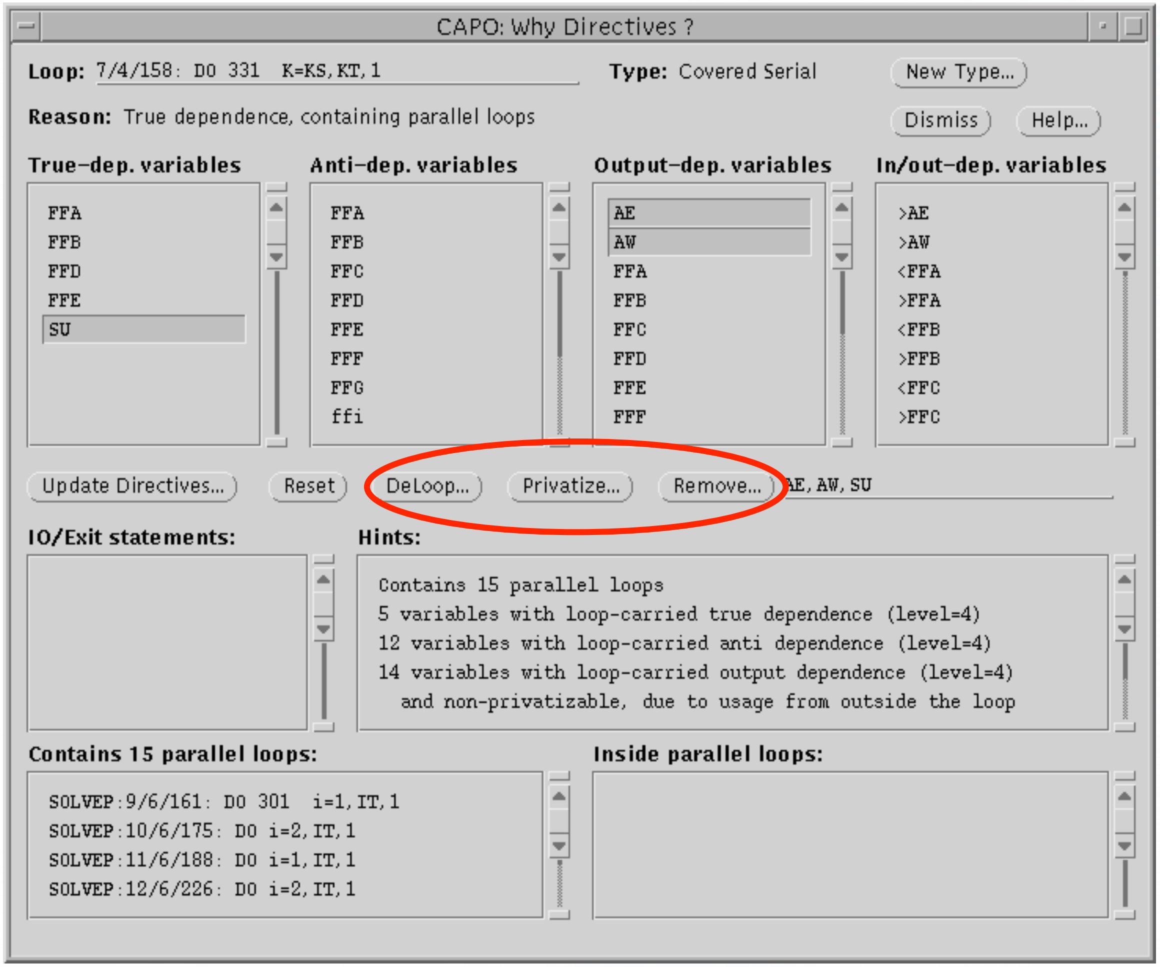
- Parallel loop that is not nested within other parallel loops
- Current Loop level at which parallel DO directive is inserted
- Includes loops identified with reduction operations
- Includes loops identified with software pipelines

Not Chosen:

- Parallel loop not chosen due to the selection of other parallel loops from the "Chosen Parallel" category above or due to I/O statements
- User may enforce parallelization if needed

The Why Directives Window

- Reason and hints for a selected loop
- List of variables and dependence types
- Tools for removing dependences



Loop: 7/4/158: DO 331 K=KS,KT,1

Type: Covered Serial

Reason: True dependence, containing parallel loops

True-dep. variables	Anti-dep. variables	Output-dep. variables	In/out-dep. variables
FFA	FFA	AE	>AE
FFB	FFB	AW	>AW
FFD	FFC	FFA	<FFA
FFE	FFD	FFB	>FFA
SU	FFE	FFC	<FFB
	FFF	FFD	>FFB
	FFG	FFE	<FFC
	ffi	FFF	>FFC

AE, AW, SU

IO/Exit statements:

Hints:

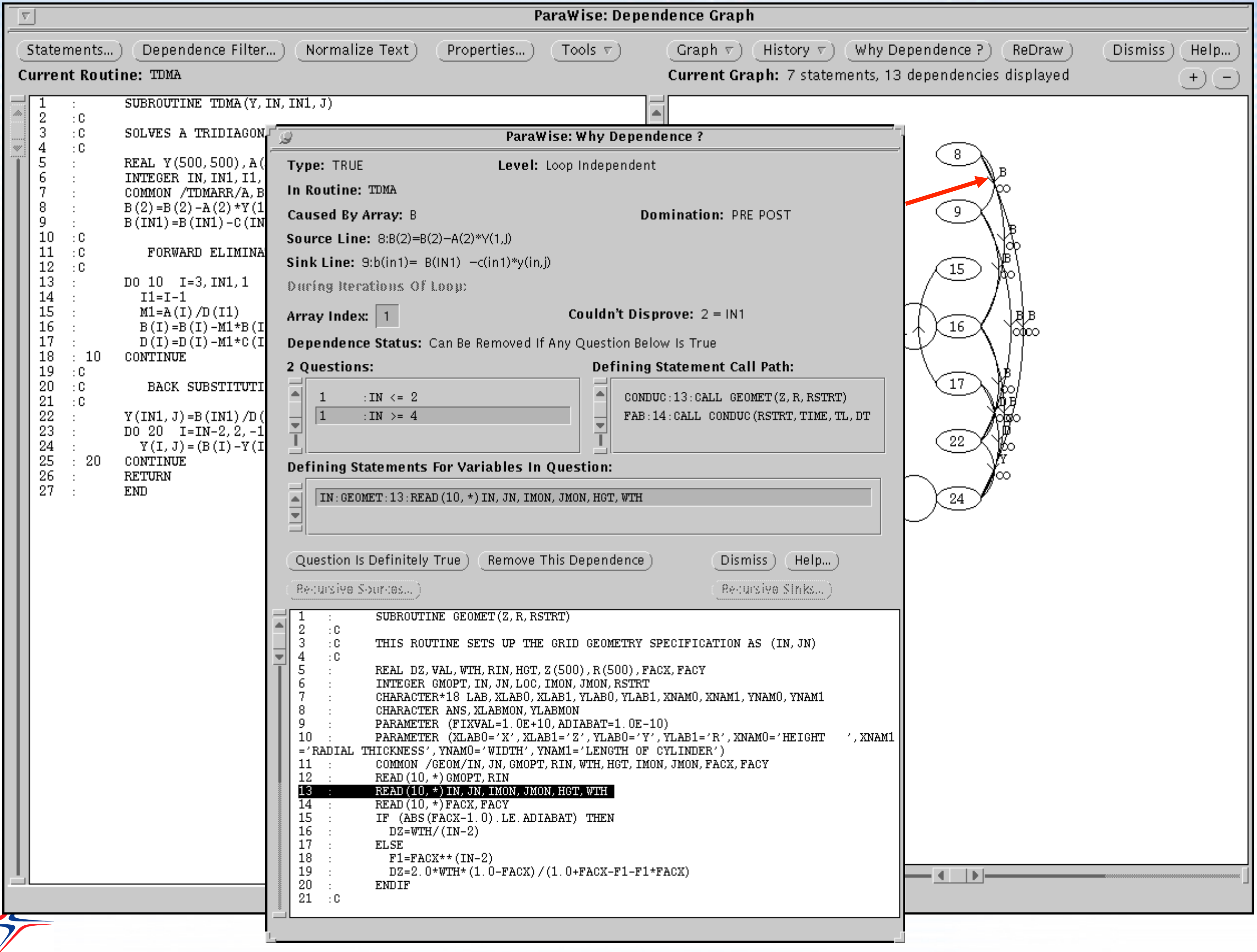
```
Contains 15 parallel loops
5 variables with loop-carried true dependence (level=4)
12 variables with loop-carried anti dependence (level=4)
14 variables with loop-carried output dependence (level=4)
  and non-privatizable, due to usage from outside the loop
```

Contains 15 parallel loops:

```
SOLVEP:9/6/161: DO 301 i=1,IT,1
SOLVEP:10/6/175: DO i=2,IT,1
SOLVEP:11/6/188: DO i=1,IT,1
SOLVEP:12/6/226: DO i=2,IT,1
```

Inside parallel loops:

Investigate Why a Dependence Is Defined



Current Routine: TDMA

Current Graph: 7 statements, 13 dependencies displayed

```
1   :       SUBROUTINE TDMA(Y, IN, IN1, J)
2   :C
3   :C      SOLVES A TRIDIAGON
4   :C
5   :       REAL Y(500,500),A(
6   :       INTEGER IN, IN1, I1,
7   :       COMMON /TDMARR/A,B
8   :       B(2)=B(2)-A(2)*Y(1
9   :       B(IN1)=B(IN1)-C(IN
10  :C
11  :C        FORWARD ELIMINA
12  :C
13  :       DO 10  I=3,IN1,1
14  :         I1=I-1
15  :         M1=A(I)/D(I1)
16  :         B(I)=B(I)-M1*B(I
17  :         D(I)=D(I)-M1*C(I
18  : 10    CONTINUE
19  :C
20  :C        BACK SUBSTITUTI
21  :C
22  :       Y(IN1,J)=B(IN1)/D(
23  :       DO 20  I=IN-2,2,-1
24  :         Y(I,J)=(B(I)-Y(I
25  : 20    CONTINUE
26  :       RETURN
27  :       END
```

ParaWise: Why Dependence ?

Type: TRUE **Level:** Loop Independent

In Routine: TDMA

Caused By Array: B **Domination:** PRE POST

Source Line: 8:B(2)=B(2)−A(2)*Y(1,J)

Sink Line: 9:b(in1)= B(IN1) −c(in1)*y(in,j)

During Iterations Of Loop:

Array Index: 1 **Couldn't Disprove:** 2 = IN1

Dependence Status: Can Be Removed If Any Question Below Is True

2 Questions:

```
1      :IN <= 2
1      :IN >= 4
```

Defining Statement Call Path:

```
CONDUC:13:CALL  GEOMET(Z,R,RSTRT)
FAB:14:CALL  CONDUC(RSTRT,TIME,TL,DT
```

Defining Statements For Variables In Question:

```
IN:GEOMET:13:READ(10,*)IN,JN,IMON,JMON,HGT,WTH
```

```
1   :       SUBROUTINE GEOMET(Z,R,RSTRT)
2   :C
3   :C      THIS ROUTINE SETS UP THE GRID GEOMETRY SPECIFICATION AS (IN,JN)
4   :C
5   :       REAL DZ,VAL,WTH,RIN,HGT,Z(500),R(500),FACX,FACY
6   :       INTEGER GMOPT,IN,JN,LOC,IMON,JMON,RSTRT
7   :       CHARACTER*18 LAB,XLAB0,XLAB1,YLAB0,YLAB1,XNAM0,XNAM1,YNAM0,YNAM1
8   :       CHARACTER ANS,XLABMON,YLABMON
9   :       PARAMETER (FIXVAL=1.0E+10,ADIABAT=1.0E-10)
10  :       PARAMETER (XLAB0='X',XLAB1='Z',YLAB0='Y',YLAB1='R',XNAM0='HEIGHT    ',XNAM1
='RADIAL THICKNESS',YNAM0='WIDTH',YNAM1='LENGTH OF CYLINDER')
11  :       COMMON /GEOM/IN,JN,GMOPT,RIN,WTH,HGT,IMON,JMON,FACX,FACY
12  :       READ(10,*)GMOPT,RIN
13  :       READ(10,*)IN,JN,IMON,JMON,HGT,WTH
14  :       READ(10,*)FACX,FACY
15  :       IF (ABS(FACX-1.0).LE.ADIABAT) THEN
16  :         DZ=WTH/(IN-2)
17  :       ELSE
18  :         F1=FACX**(IN-2)
19  :         DZ=2.0*WTH*(1.0-FACX)/(1.0+FACX-F1-F1*FACX)
20  :       ENDIF
21  :C
```

Further Code Optimization

- Choose outer-most loops for better granularity

- Prune data dependences when
  - unknown information involved (e.g. input parameters)
  - code too complicated (e.g. FFT)

- Require user knowledge

- Use dialog boxes in the WhyDirectives window
  - remove ***false*** data dependences
  - thus parallelize a loop

Remove Data Dependences

```
S1      integer indexptr(maxcells)
      read*,indexptr
      do i=1,ncells
S1      u(indexptr(i)) = . . .
S2               . . . = u(indexptr(i)) + . . .
      enddo
S3    print*,(u(j),j=1,ncells)
```

analysis

- i loop serial due to loop carried pseudo dependences of u, $S_1 \rightarrow S_1$ (output), $S_2 \rightarrow S_1$ (anti), Loop output $S_1 \rightarrow S_3$ also u is not PRIVATE

user inspection

- Examine Loop output dependence and determine it is correct therefore u cannot be PRIVATE

possible solution

- If contents of indexptr are all unique then we can safely remove the loop carried anti and output dependencies for the array u allowing u to stay SHARED and the loop to execute in parallel

Remove Data Dependences (cont.)

```
S1    read*,(work(k),k=1,10)
      do i=1,10
        do j=1,n
S2        work(j)=j
        enddo
S3      b(i)=b(i)+work(2)
      enddo
```

analysis

- `i` loop serial due to loop carried pseudo dependencies of `work`, $S_2 \rightarrow S_2$ (output), $S_3 \rightarrow S_2$ (anti)
- Loop input dependence of `work`, $S_1 \rightarrow S_3$ (true) exists so `work` is not `PRIVATE`

user inspection

- Examine in Why dependence window of dependence graph browser
- Determine that the pseudo dependencies are correct (`work` is re-used)
- Loop input dependence non-existent if `n >=2`

possible solution

- Delete loop input dependence or (preferably) add `n>=2` to info + re-analyze. `work` is now `PRIVATE` and `i` loop can execute in parallel

Remove Data Dependences (cont.)

```
S1    read*,(work(k),k=1,10),(n(k),k=1,10)
      do i=1,10
        do j=1,n(i)
S2        work(j)=j
        enddo
S3      b(i)=b(i)+work(2)
      enddo
```

analysis

- Now `n` is an array – additional true dependence of work carried by `i` loop $S_2 \rightarrow S_3$
- `i` loop appears to be inherently serial

user inspection

- Examine true dependence first, others only important if it can be removed
- Loop carried true dependence non-existent if all `n(1:10) >= 2`

possible solution

- Delete loop carried true dependence followed by loop input dependence (as before) or just add `n(1:10) >= 2` to info + re-analyze
- `i` loop is now parallel and `work` is `PRIVATE`

NAS NASA Advanced Supercomputing

Remove Data Dependences (cont.)

```
   do k=1,d(3)
    do j=1,d(2)
     do i=1,d(1)
S1    y1(j,i)=. . .
     enddo
    enddo
S2  call cfftz(y1, . . .)
    do j=1,d(2)
     do i=1,d(1)
S3    . . . = y1(j,i)
     enddo
```

analysis

- `k` loop is apparently serial since `y1` is assigned in S_1 and S_2 and is used in S_2 and S_3 i.e. true dependence $S_2 \rightarrow S_2$

user inspection

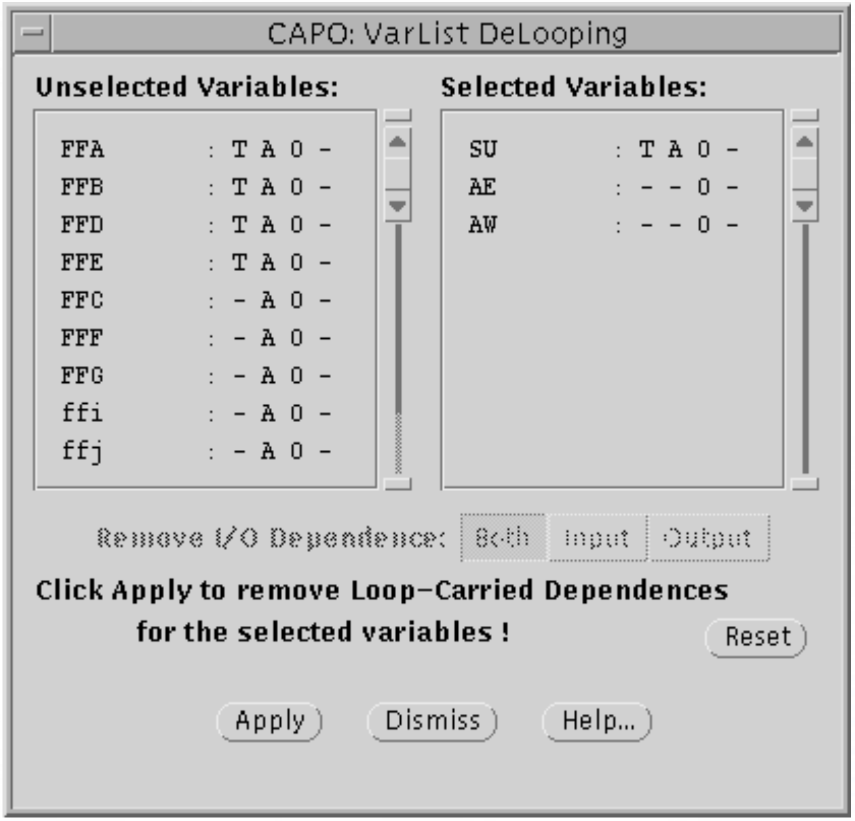
- Examine true dependence first, others only important if it can be removed.
- Examine loop Input/Output dependence

possible solution

- If it is known that there are no assignments of `y1` before S_1 then we can safely remove the loop carried true dependences and Input/Output dependences for `y1` making it

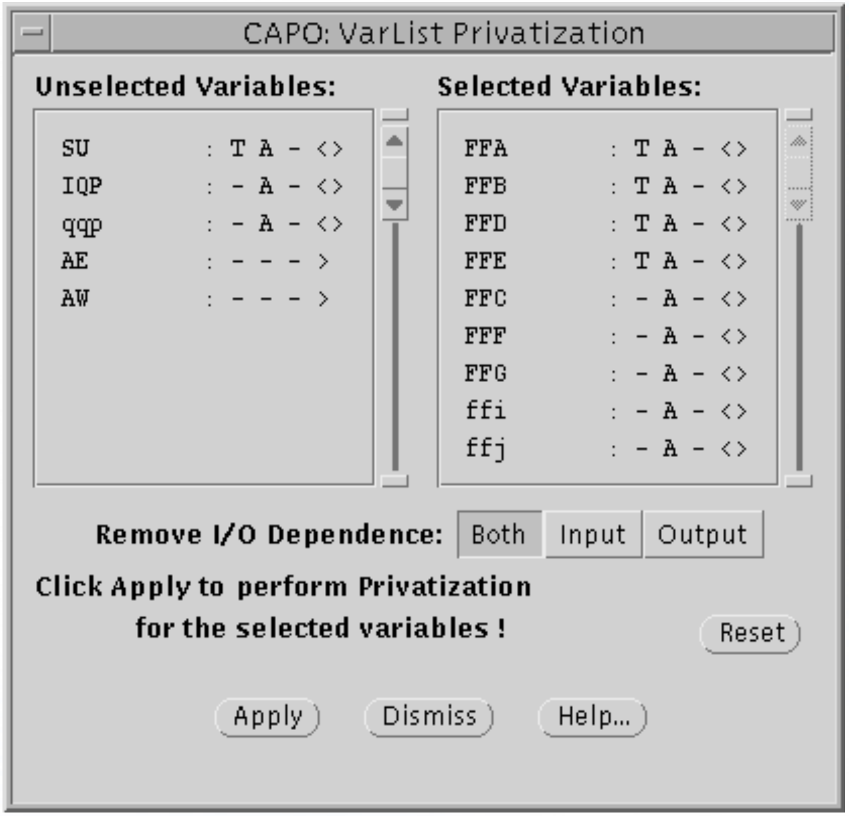
Remove Data Dependences (cont.)

- [*DeLoop*]
  - make variables ***shared*** → delete ***loop-carried*** dependences
- [*Privatize*]
  - make variables ***private*** → delete loop-carried ***True/Anti*** dependences and ***Input/Output*** True dependences



```
FFA    : T A O -        SU    : T A O -
FFB    : T A O -        AE    : - - O -
FFD    : T A O -        AW    : - - O -
FFE    : T A O -
FFC    : - A O -
FFF    : - A O -
FFG    : - A O -
ffi    : - A O -
ffj    : - A O -
```

Click Apply to remove Loop-Carried Dependences
for the selected variables !

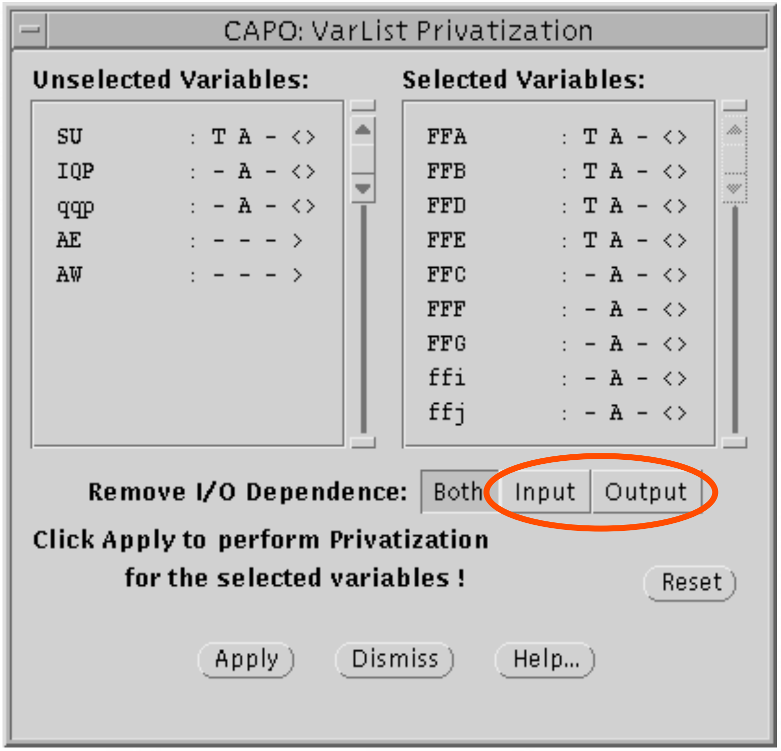


```
SU     : T A - <>       FFA    : T A - <>
IQP    : - A - <>       FFB    : T A - <>
qqp    : - A - <>       FFD    : T A - <>
AE     : - - - >        FFE    : T A - <>
AW     : - - - >        FFC    : - A - <>
                        FFF    : - A - <>
                        FFG    : - A - <>
                        ffi    : - A - <>
                        ffj    : - A - <>
```

Click Apply to perform Privatization
for the selected variables !

Remove Data Dependences (cont.)

- [*Privatize*] continued
  - It is possible to make variables ***firstprivate*** or ***lastprivate*** → select "Remove ***Output*** (**>**) or ***Input*** (**<**) dependences"



CAPO: VarList Privatization

Unselected Variables:

```
SU      : T A - <>
IQP     : - A - <>
qqp     : - A - <>
AE      : - - - >
AW      : - - - >
```

Selected Variables:

```
FFA     : T A - <>
FFB     : T A - <>
FFD     : T A - <>
FFE     : T A - <>
FFC     : - A - <>
FFF     : - A - <>
FFG     : - A - <>
ffi     : - A - <>
ffj     : - A - <>
```

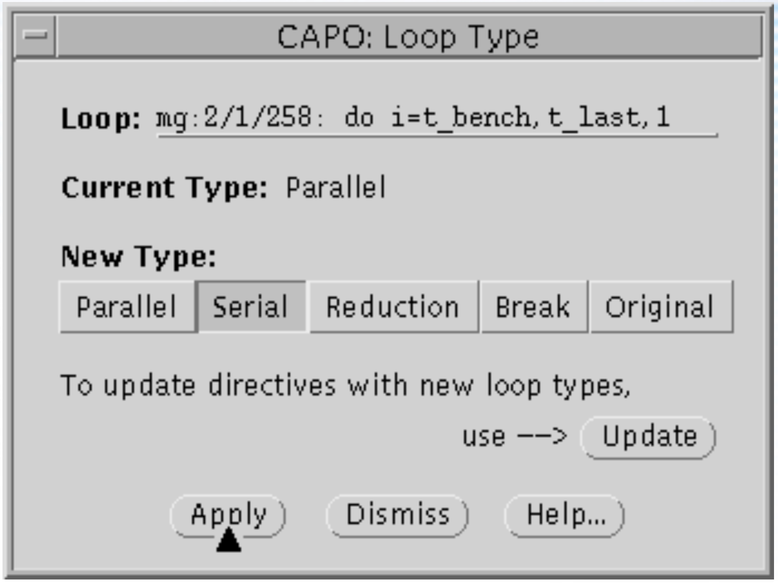
Remove I/O Dependence:

Click Apply to perform Privatization for the selected variables !

Caution! User can improve performance ***but also*** can introduce mistakes

Further Optimization

- User enforced loop types
  - overwrite a default
    - for I/O loops
    - concerning granularity
  - use the Loop Type window



CAPO: Loop Type

Loop: mg:2/1/258: do i=t_bench,t_last,1

Current Type: Parallel

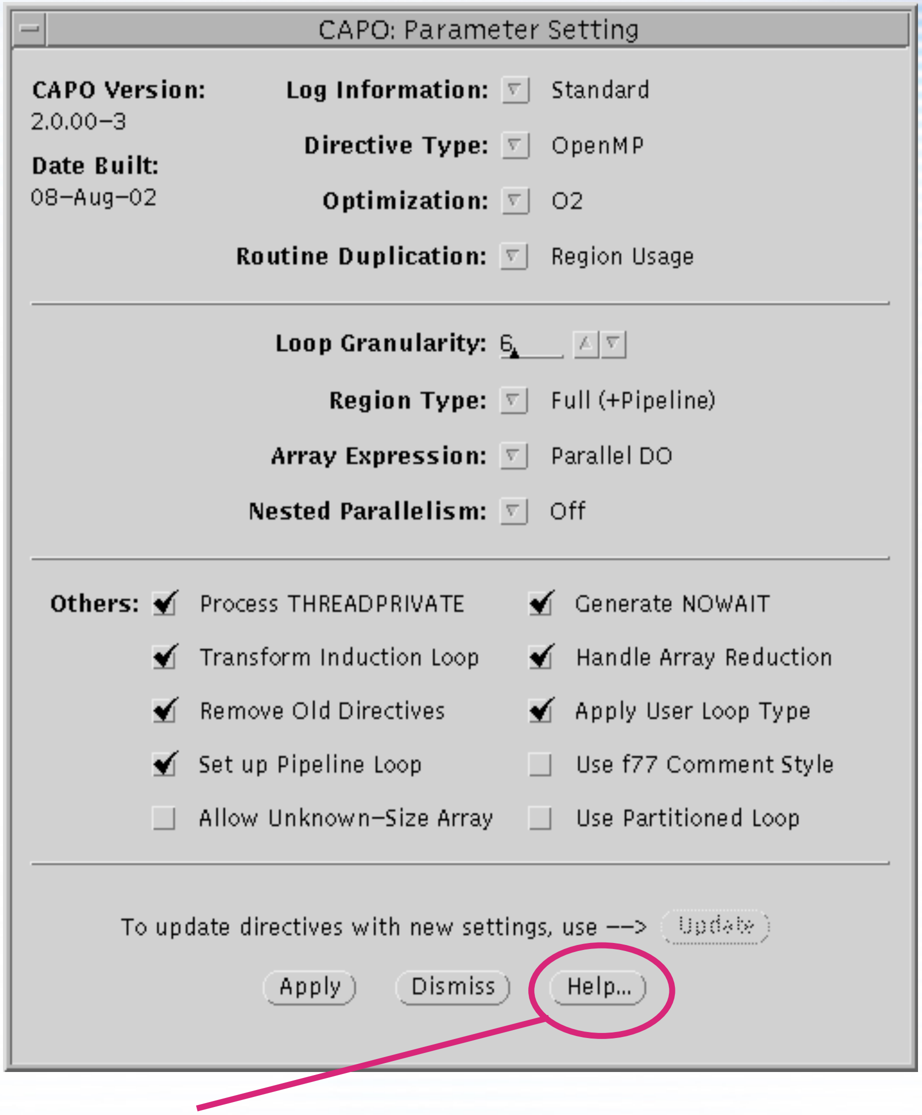
New Type:

To update directives with new loop types,
use --> Update

- The “`userloop.par`” file
  - User defined loop types are saved to this file, read back automatically from the file
  - A different filename may be specified via the environment variable **`CAPO_USERLOOP`**

Parameters to Control the CAPO Execution

- Setting dialog box
  - set most parameters
- Environment variables
  - GUI correspondence
    - **`CAPO_LOGINFO`**
    - . . .
  - no GUI correspondence
    - **`CAPO_USERLOOP`**
    - . . .



CAPO: Parameter Setting

CAPO Version: 2.0.00-3

Date Built: 08-Aug-02

Log Information: Standard

Directive Type: OpenMP

Optimization: O2

Routine Duplication: Region Usage

Loop Granularity: 6

Region Type: Full (+Pipeline)

Array Expression: Parallel DO

Nested Parallelism: Off

Others:

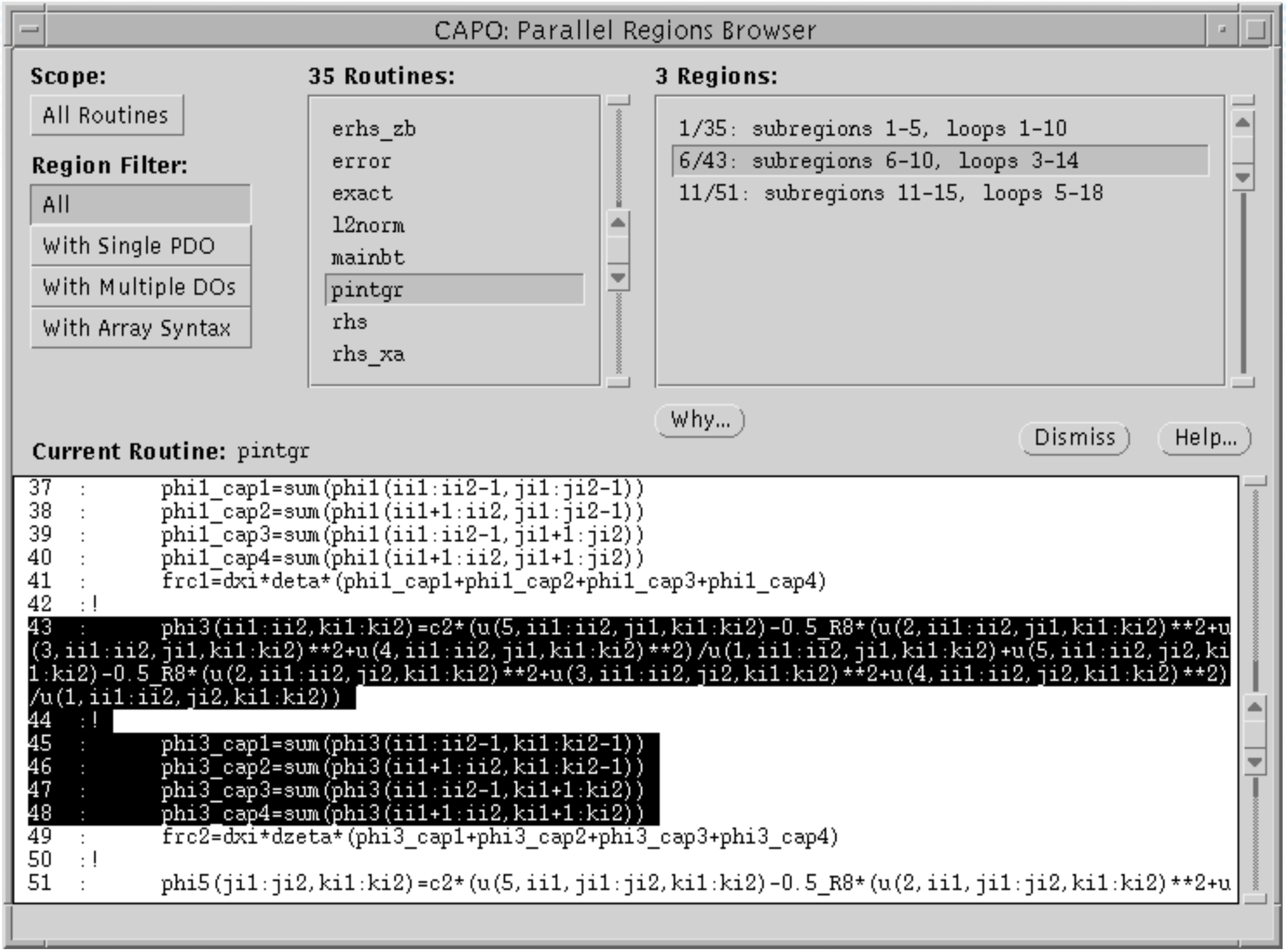
- ☑ Process THREADPRIVATE
- ☑ Transform Induction Loop
- ☑ Remove Old Directives
- ☑ Set up Pipeline Loop
- □ Allow Unknown-Size Array
- ☑ Generate NOWAIT
- ☑ Handle Array Reduction
- ☑ Apply User Loop Type
- □ Use f77 Comment Style
- □ Use Partitioned Loop

To update directives with new settings, use --> Update

if you are not sure

Browsing Parallel Regions

- With the *Parallel Regions* browser



Scope: All Routines

Region Filter:

- All
- With Single PDO
- With Multiple DOs
- With Array Syntax

35 Routines:

```
erhs_zb
error
exact
l2norm
mainbt
pintgr
rhs
rhs_xa
```

3 Regions:

```
1/35:  subregions 1-5,  loops 1-10
6/43:  subregions 6-10,  loops 3-14
11/51:  subregions 11-15,  loops 5-18
```

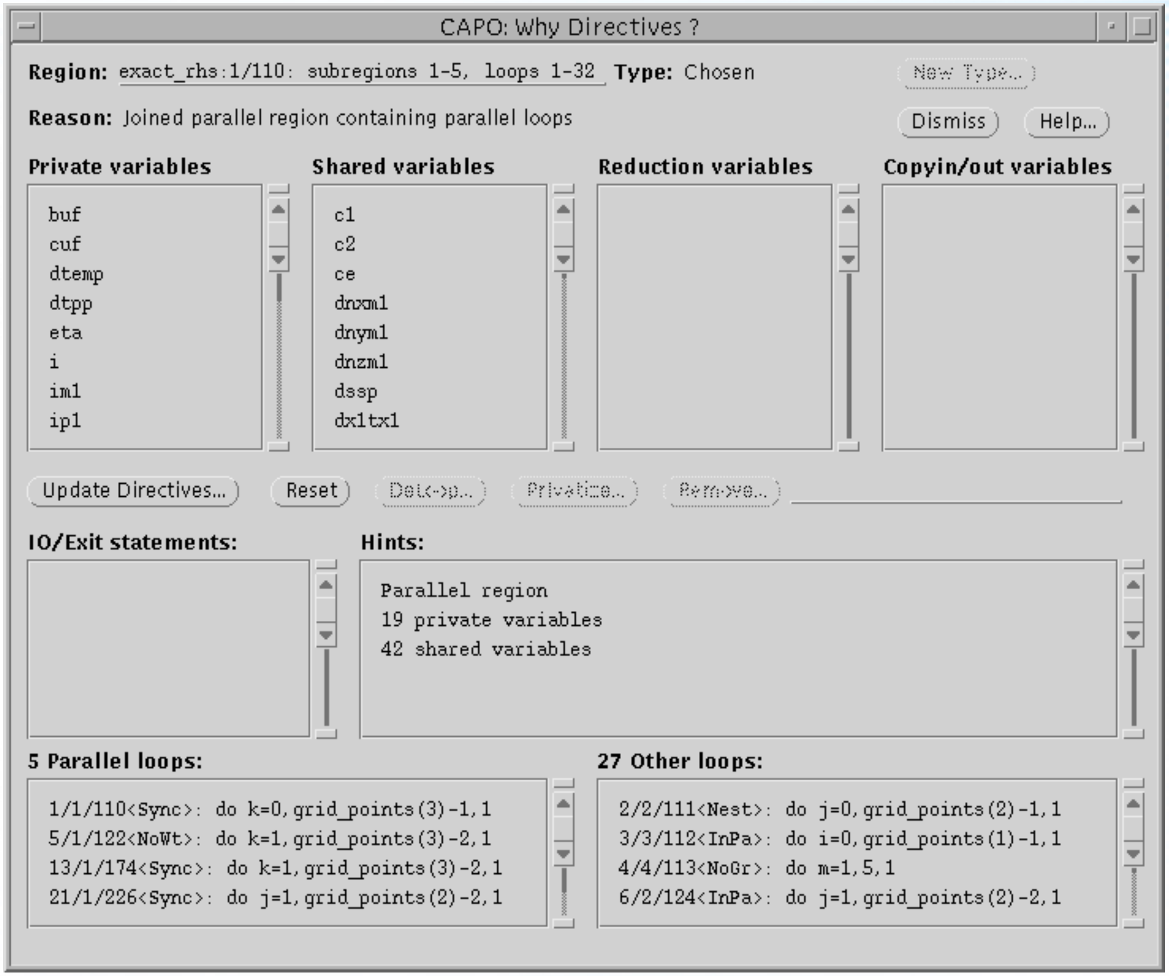
Current Routine: pintgr

```
37   :       phi1_cap1=sum(phi1(ii1:ii2-1,ji1:ji2-1))
38   :       phi1_cap2=sum(phi1(ii1+1:ii2,ji1:ji2-1))
39   :       phi1_cap3=sum(phi1(ii1:ii2-1,ji1+1:ji2))
40   :       phi1_cap4=sum(phi1(ii1+1:ii2,ji1+1:ji2))
41   :       frc1=dxi*deta*(phi1_cap1+phi1_cap2+phi1_cap3+phi1_cap4)
42   :!
43   :       phi3(ii1:ii2,ki1:ki2)=c2*(u(5,ii1:ii2,ji1,ki1:ki2)-0.5_R8*(u(2,ii1:ii2,ji1,ki1:ki2)**2+u
(3,ii1:ii2,ji1,ki1:ki2)**2+u(4,ii1:ii2,ji1,ki1:ki2)**2)/u(1,ii1:ii2,ji1,ki1:ki2)+u(5,ii1:ii2,ji2,ki
1:ki2)-0.5_R8*(u(2,ii1:ii2,ji2,ki1:ki2)**2+u(3,ii1:ii2,ji2,ki1:ki2)**2+u(4,ii1:ii2,ji2,ki1:ki2)**2)
/u(1,ii1:ii2,ji2,ki1:ki2))
44   :!
45   :       phi3_cap1=sum(phi3(ii1:ii2-1,ki1:ki2-1))
46   :       phi3_cap2=sum(phi3(ii1+1:ii2,ki1:ki2-1))
47   :       phi3_cap3=sum(phi3(ii1:ii2-1,ki1+1:ki2))
48   :       phi3_cap4=sum(phi3(ii1+1:ii2,ki1+1:ki2))
49   :       frc2=dxi*dzeta*(phi3_cap1+phi3_cap2+phi3_cap3+phi3_cap4)
50   :!
51   :       phi5(ji1:ji2,ki1:ki2)=c2*(u(5,ii1,ji1:ji2,ki1:ki2)-0.5_R8*(u(2,ii1,ji1:ji2,ki1:ki2)**2+u
```

NAS
NASA Advanced Supercomputing

Browsing Parallel Regions (cont.)

- Connection to the WhyDirectives window
  - list of variables and their types
  - indication of the end-of-loop synchronization
- No direct modification to regions



CAPO: Why Directives ?

Region: exact_rhs:1/110: subregions 1-5, loops 1-32 **Type:** Chosen

Reason: Joined parallel region containing parallel loops

Private variables

```
buf
cuf
dtemp
dtpp
eta
i
im1
ip1
```

Shared variables

```
c1
c2
ce
dnxm1
dnym1
dnzm1
dssp
dx1tx1
```

Reduction variables

Copyin/out variables

IO/Exit statements:

Hints:

```
Parallel region
19 private variables
42 shared variables
```

5 Parallel loops:

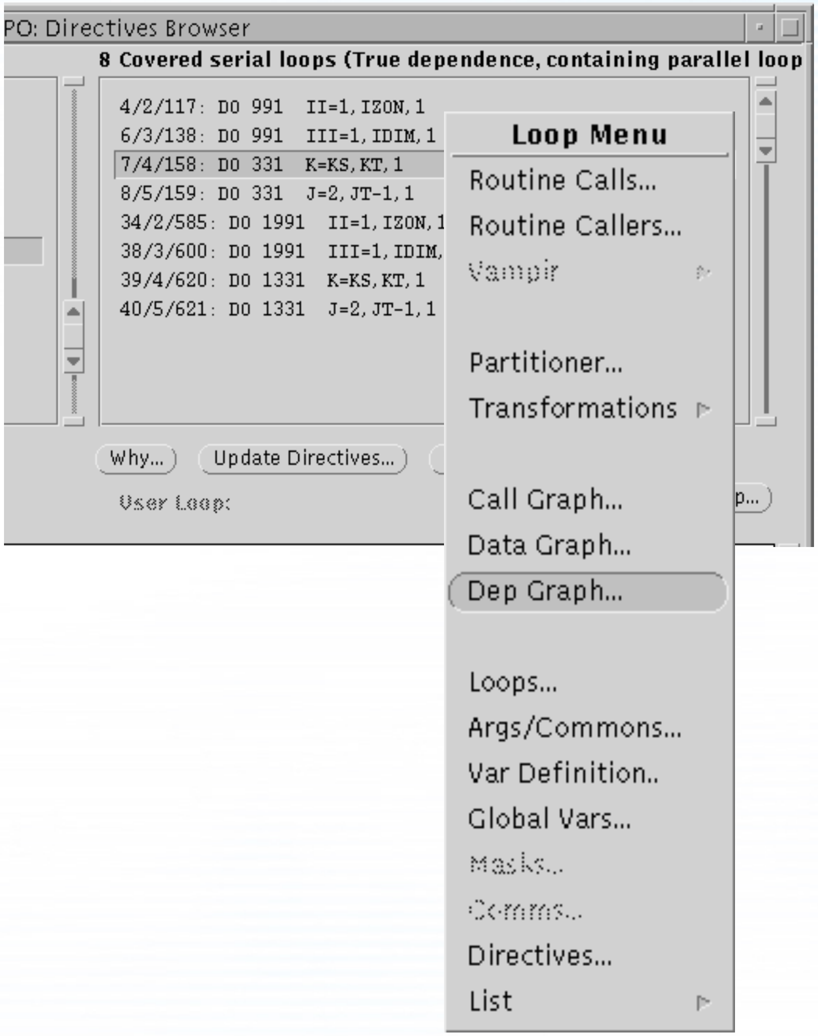
```
1/1/110<Sync>:  do k=0,grid_points(3)-1,1
5/1/122<NoWt>:  do k=1,grid_points(3)-2,1
13/1/174<Sync>:  do k=1,grid_points(3)-2,1
21/1/226<Sync>:  do j=1,grid_points(2)-2,1
```

27 Other loops:

```
2/2/111<Nest>:  do j=0,grid_points(2)-1,1
3/3/112<InPa>:  do i=0,grid_points(1)-1,1
4/4/113<NoGr>:  do m=1,5,1
6/2/124<InPa>:  do j=1,grid_points(2)-2,1
```

Hotlinks

- Quick access to other functions
- Menus from pressing the *right* mouse button
  - linked with a loop
  - linked with a variable
  - linked with a routine
  - linked with a textline
- Example
  - bring up the DepGraph window for the selected loop



PO: Directives Browser

8 Covered serial loops (True dependence, containing parallel loop

```
4/2/117:  DO 991   II=1,IZON,1
6/3/138:  DO 991   III=1,IDIM,1
7/4/158:  DO 331   K=KS,KT,1
8/5/159:  DO 331   J=2,JT-1,1
34/2/585:  DO 1991   II=1,IZON,1
38/3/600:  DO 1991   III=1,IDIM,
39/4/620:  DO 1331   K=KS,KT,1
40/5/621:  DO 1331   J=2,JT-1,1
```

Command Interface for the Batch Mode

- Provide access to the functionality of GUI components without starting the GUI
- Commands usually recorded to a command file by

```
capo –logfile capo_run.cmd
```

- Played back [in a batch mode] with

```
capo [-batch] capo_run.cmd
```

- Commands in the command interface are given in the *CAPO User Manual* A4

Hybrid Parallelization

- Existing message passing codes
  - Use CAPO to insert OpenMP directives
- Two-step process
  - First: ParaWise to generate the message-passing code
  - Second: CAPO to insert OpenMP directives
- Issues
  - No communication routines allowed inside a parallel region
  - The partitioned dimension is not used for OMP loop level parallelization, but it is possible to enforce the choice
    - In the Setting Box, check "Use Partitioned Loop"
- See an example in the CAPO tutorial notes

Fortran 90/95 Codes

- In the beta stage
- Main feature – handling array syntax, **FORALL** loop, **WHERE** construct
  - convert to a regular DO loop
  - use "**OMP WORKSHARE**" (not yet supported)
  - do nothing, let a compiler work it out

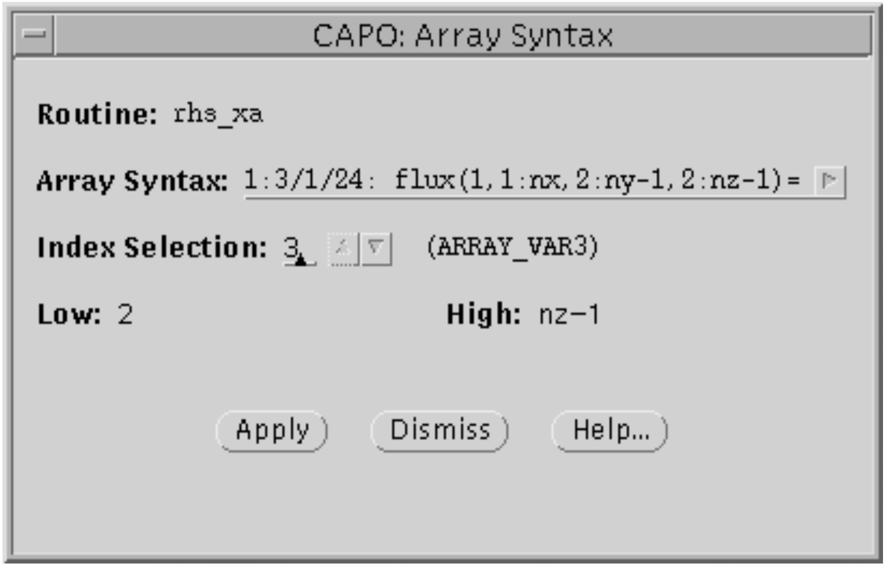
```
 flux(2,2:nx-1,2:ny-1,2:nz)=tz3*(du2(2:nx-1,2:ny-1,2:nz) &
&                             -du2(2:nx-1,2:ny-1,1:nz-1))
```

converted to

```
 do ARRAY_VAR3=2,nz
   flux(2,1:nx,2:ny-1,ARRAY_VAR3)=tz3*                    &
&                   (du2(2:nx-1,2:ny-1,ARRAY_VAR3)        &
&                   -du2(2:nx-1,2:ny-1,ARRAY_VAR3-1))
```

Fortran 90/95 Codes (cont.)

- Control of the array syntax conversion
  - done automatically
  - user can overwrite:
    select an index dimension for conversion



CAPO: Array Syntax

Routine: rhs_xa

Array Syntax: 1:3/1/24: flux(1,1:nx,2:ny-1,2:nz-1)=

Index Selection: 3 (ARRAY_VAR3)

Low: 2 High: nz-1